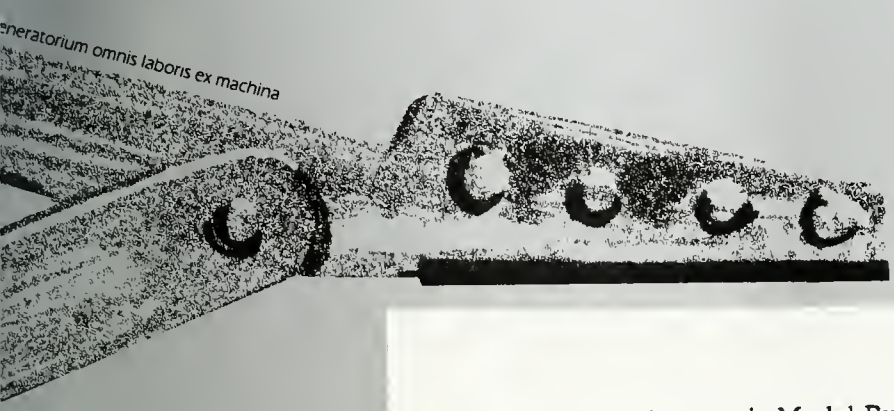
Robotics Research Technical Report

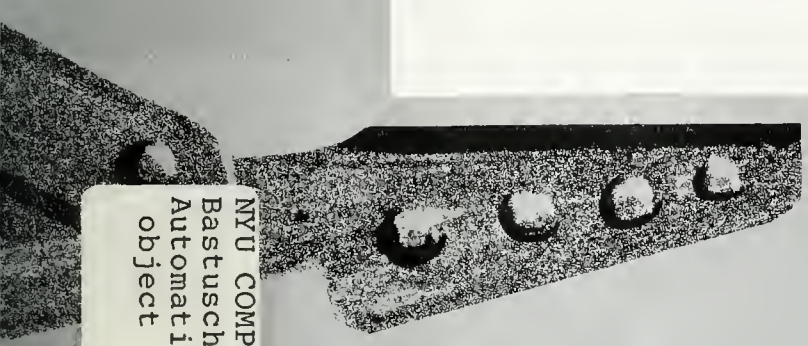


Automatic Model Builder
for Object Recognition

by

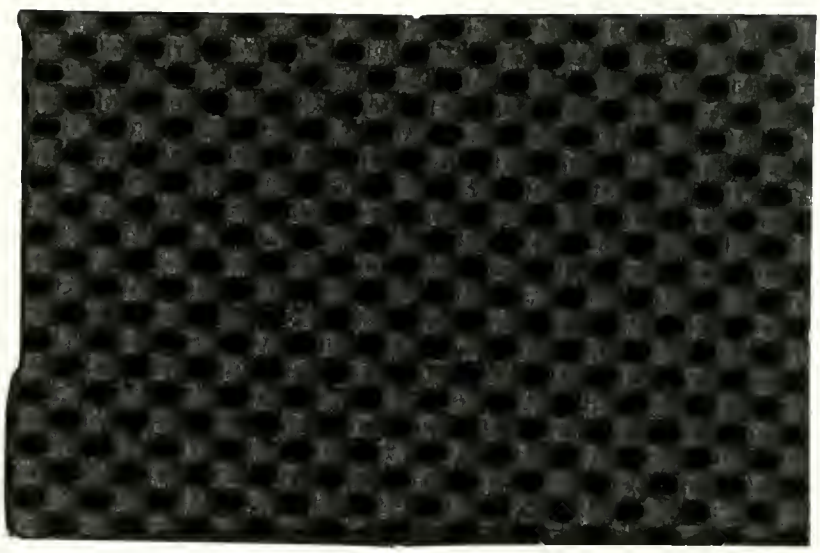
C. Marc Bastuscheck

Technical Report No. 321
Robotics Report No. 121
September, 1987



New York University
t Institute of Mathematical Sciences

Computer Science Division
251 Mercer Street New York, N.Y. 10012



Automatic Model Builder for Object Recognition

by

C. Marc Bastuscheck

Technical Report No. 321
Robotics Report No. 121
September, 1987

New York University
Dept. of Computer Science
Courant Institute of Mathematical Sciences
251 Mercer Street
New York, New York 10012

Work on this paper has been supported by Office of Naval Research Grant N00014-82-K-0381, National Science Foundation CER Grant DCR-83-20085, and by grants from the Digital Equipment Corporation and the IBM Corporation.

Automatic Model Builder for Object Recognition[1]

C. Marc Bastuscheck

Robotics Activity
Courant Institute
New York University

ABSTRACT

Automatic construction of 3D models for use in object recognition is considered. A model builder is described that extracts lines and feature points from sets of range and intensity images of an object, determines which features in one view correspond to which features in other views, determines the transformations between successive views, and constructs a single model from appropriate overlapping views. A complete and accurate model of a decorated flowerpot was constructed from 9 views using only data provided by an imaging range sensor, which is described. The procedure required about 1 minute on a VAX 785.

1. Introduction

Object recognition by machine consists of data acquisition, data reduction such as feature extraction or region segmentation, and comparison of processed data with a suitable model. There has been considerable effort expended in developing sensors and techniques for gathering data, and also in developing procedures for processing data. There has been relatively little work on developing models. Of course, with two dimensional objects a previous image can serve as a model, and it has been possible to describe three dimensional objects using a number of two dimensional projections. However, true three dimensional models appear to have a number of advantages for use with robot vision systems. In this paper a method is described for generating a model that contains accurate 3D information about an observed object. A particular implementation is described and analyzed.

There is need for an automatic model building procedure. At present models are often simply previous, processed views of objects. This works well for 2D objects, or when an object consists of a single line in 3-space [BSSS86], but less well when full three-dimensional objects are used, since many models of the same object are needed to cover all possible views [OS83]. For manufactured parts CAD descriptions may be available that can be used for feature based object recognition [BHH83],[HB84], or such descriptions can be created off-line [Kn87]; however, CAD descriptions may not be available for arbitrary objects or may require extensive processing to make them compatible with information which can be derived by a vision system. A technique is described by which dense data from a 3D sensing system can be combined to generate a complete 3D model of an observed object. Such a model would be optimum for use with the vision system which supplied the

[1]Work on this paper has been supported by Office of Naval Research Grant N00014-82-K-0381, National Science Foundation CER Grant DCR-83-20085, and by grants from the DEC and IBM corporations.

data, but could have much wider application. For example, there is considerable interest in sensing systems that identify objects based on sparse data [GL84],[Ko86] or analysis of 2D images [L85],[TM87]; these require a 3D model of an object but cannot generate a suitable model.

The primary difficulty encountered in constructing a model of a three-dimensional object is combining the multiple views necessary to capture all sides of it. A simple approach is to make dense range measurements of the object in a number of known orientations, e.g. by placing the object on a turntable and recording the rotation between successive images [VA86],[CMST87]. Potmesil [P83] formed a complete surface representation from many images of an object using a computer guided iterative search to determine optimum transformations between known overlapping views. Faugeras and his co-workers gathered complete data from their auto part in known orientations [Fea83], but also developed a technique of solving for the transformation [FH83] needed to bring one surface into best juxtaposition with a second surface; they have identified overlapping views of an object by comparing segmented surface patches of the observed object with similar descriptions of a complete model.

The model builder described here is based on the use of lines and feature points for object recognition. Description of an object in terms of clearly determined lines and points reduces uncertainty of object location and orientation. When two views of an object contain a common region data from the views can be combined by using data from the regions of overlap to determine the transformation that brings one view into the coordinate frame of the other. This procedure can be repeated to bring a complete set of views of an object into a single frame of reference. The amplification of this idea into a model builder is described generally in section 2. In section 3 a model builder is described that combined data from multiple registered range and intensity views of a decorated flowerpot. The system used only information derived from these views, and had no *a priori* knowledge of relations between the various views. Extension and application of this system is described in section 4. An appendix describes algorithms and data structures in detail.

2. A Model Builder ...

A model builder should generate a single complete model of an object. Model building includes data acquisition and reduction, combination of multiple views to a single frame of reference, and formation of an appropriate model from combined data. Multiple views are required since a three dimensional object cannot be adequately observed in a single view, and combination of multiple views into a single frame of reference is central to the model builder. While it is possible to record data from an object in known orientations, this can be cumbersome in practice, and a model builder should not require such assistance. The specification of what constitutes an appropriate model depends on the intended use and sensing system, and many different descriptions can be developed from one model and original data. For example, the model builder of section 3 is based on feature points, but the final product is a model that describes an object in terms of lines, and a similar extension could be made to generate a surface model.

In this section the actions of an ideal model builder are described, both generally and with some suggestions of specific implementations. In the following section an actual implementation is described.

Extracts suitable information from observations. Any observations that yield reliable 3D coordinates can be used as the basis for an automatic model builder. Among the many features that may be used are lines significant to object shape, feature points such as vertices, or identifiable surface patches. Extracted features have three dimensional coordinates, but since the object can be moved arbitrarily between observations a particular feature will have different coordinates in each set of observations.

Uses features to find corresponding regions in pairs of views. Data collection may be structured so it is known which views of an object contain common regions, and views can be matched immediately. Often, it is necessary to determine views which contain overlapping regions. Features, e.g. the perimeter of a paisley-shaped area of different reflectivity, which may be easily identified in several views, should be exploited. Alternatively, relations between features, such as angles and distances between line segments, might be used to create a list of views ordered in probability of overlap, and also information concerning which features of one view are likely to correspond to features in another.

Matches overlapping views successively. A single model is constructed by sequentially matching and transforming the coordinates of many views. For example, suppose that views A,B,C,... are made of an American football rotated about its axis 20 or 30 degrees between each observation. (The exact amount of rotation is immaterial, since the model-builder will discover the proper transformation between all sets of observations it uses; similarly the axis need not be constant.) View B may be found to have regions of overlap with views A, C, and D. A good matching order would be B, C, D, ..., A, where the coordinate transformation to bring C into the frame of reference of B would be found, C would be transformed using this, then the transformation needed to bring D into this frame of reference would be found, and so on. The model builder should distinguish between this order (B,C,D,...,A) and less favorable orders such as B, A, C, D ... that would be possible if the amount of rotation between observations were small.

Recognizes closed cycles of overlapping views. In the simple football example above the model builder would not know *a priori* order of observations or that the object was rotated about its axis. However, if it starts its successive match chain with view B, C, ... it should recognize when it gets to A that it has a complete a cycle of views. Objects may have several closed cycles of views, and these should be taken into account when the optimum order of matching is determined. It may also be possible to present objects to the model builder in such a manner as to eliminate multiple closed cycles: for example, a cube could be rotated about a long diagonal instead of about an axis perpendicular to a face.

Resolves conflicts after match-around. There is uncertainty in data from observations, resulting in error in each of the transformations, and this error may accumulate during successive matches, resulting in the displacement of the final view from where it would be if it were matched directly to the first view. Systematic errors in the range sensor may cause error in successive matches to combine coherently rather than in a random manner, resulting in large errors. A coherent

model can be built only if match-around error is negligible or can be resolved.

Creates list of unique features in a single coordinate frame. Most features of the object will be observed in several views, but a single feature will not appear at exactly the same location in each of its transformed views because of errors in observation and transformation. The model builder must determine an appropriate location for each separate feature. It should not be confused by distinct features that are similar.

Finishes the model. A list of unique features in a common coordinate system constitutes a model of the object. However, features in earlier partial processing or the original data may be transformed (using the transformations generated in the match-around process) to yield a different model, such as a complete surface/reflectivity description of the object. Additional processing may be desired to generate tables used in object recognition.

Works interactively, intermittently, or in batch mode to make complete model. In some circumstances it is desirable to generate a complete model of an object. An operator (or perhaps robot, in an advanced implementation) places an object repeatedly in the work area of a vision sensor so overlapping images of all sides are obtained. In batch mode the operator would decide when appropriate images had been made. The model builder might prompt the operator if it needed additional views. In interactive mode the model builder would prompt the operator to place the object in a new approximately known position. Alternatively, it may not be important that an object be completely known, but a robot vision system might update its model of a peripheral object as new regions became visible during normal workcell operations.

3. A first implementation

To demonstrate feasibility of automatic model building, a system of hardware and software was assembled that generated a model of a decorated flowerpot from multiple intensity and range images. Multiple views of the object were made, features were extracted, views having overlapping regions were identified and successively matched to form a complete model. The position of the object in each view was not recorded except by the imaging range sensor, and the model-building program did not know *a priori* which views would or would not contain overlapping regions. The implemented model builder was simpler than the general model builder of section 2. For example, the algorithms used only one type of feature (one not previously used in object recognition studies), and the object required only one closed cycle of views to represent it fully.

3.1. The Problem

The object used in this demonstration was a terracotta flowerpot approximately 17 cm in diameter at the top and 15 cm high having 17 disks of yellow construction paper glued to the conical side surface. The paper disks, spaced irregularly about the pot, were all 3.5 cm in diameter, and were probably indistinguishable. The perimeters of the disks in 3-space were extracted from registered intensity and range

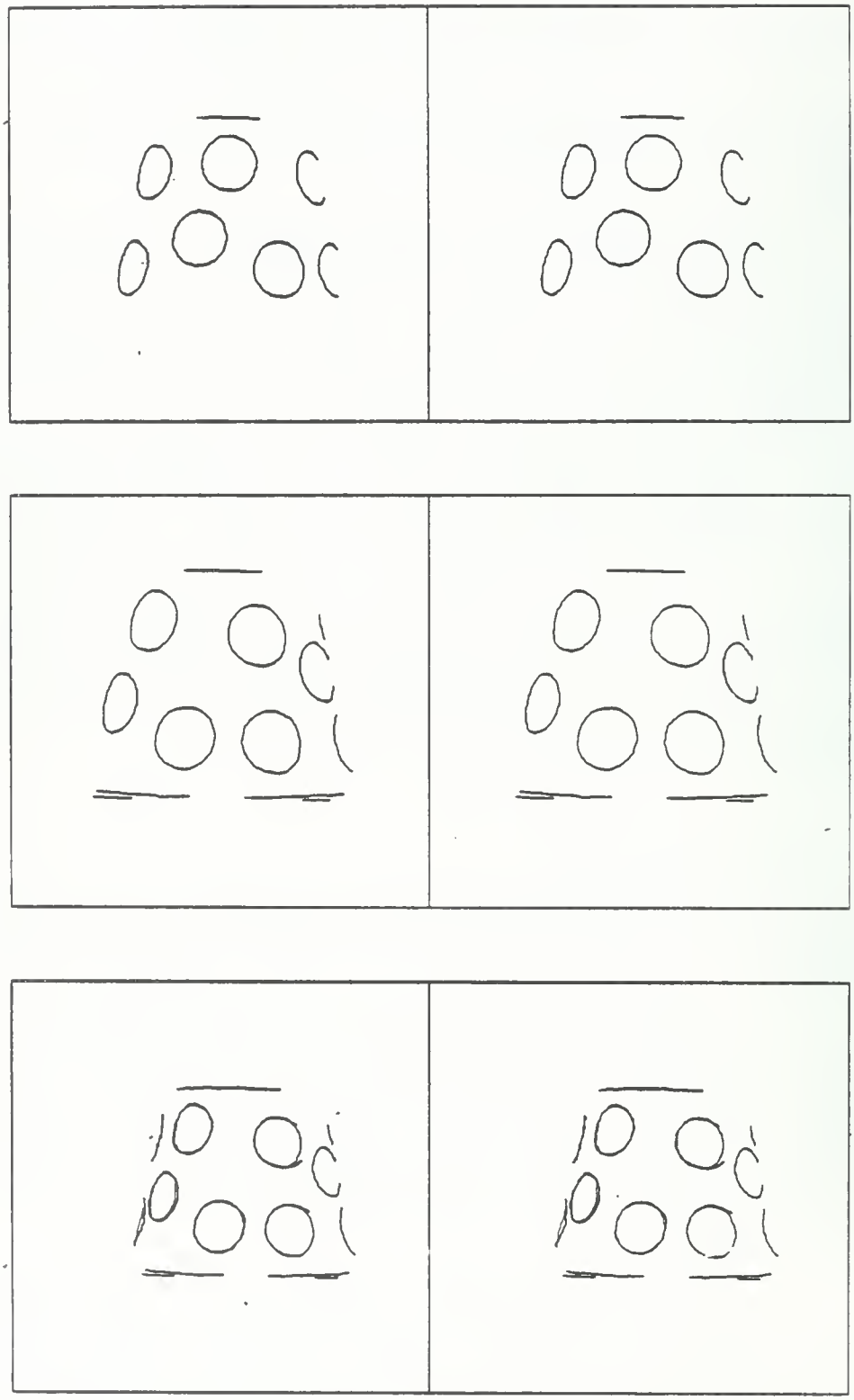


Figure 1. Views 4 (top) and 6 (middle), and both views after matching (bottom). Look at left stereoimage with left eye, right image with right.

images of the object using established procedures[BSSS86]. Some lines corresponding to edges and the lip of the pot were also extracted incidentally; these were not removed from the data, but were probably not used in the matching process. Fig. 1 shows two sets of typical data, and how they appear after matching. The curves used in matching are simply data points, and are not parametrized in any way.

Features used for matching were points of closest approach of the extracted curves. To reduce the effects of noise and quantization, points of closest approach were defined to be the average of a number of points on both sides of the actual point of closest approach[B86]. It would have been possible to fit circles to the curves and use the centers of the circles as features, but points of closest approach were both more general and theoretically a better description of the data, since the edges of the disks on the conical surface were not circles. In a more fully developed model builder it is expected that additional types of lines including geometric edges or lines of maximum curvature would be identified and extracted, and that a number of different features would be exploited. However, this new feature, points of closest approach of two curves in space, worked well.

3.2. Acquisition of Range Data

The range sensor used in these experiments is an improved version of an NYU ratio image sensor described previously[BS84][BS86]. The sensor is based on optical triangulation, using a video camera to view a work area while a slide projector laterally offset from the camera illuminates the work area successively using two different filters. One filter increases in optical density from left to right while the other increases from right to left (with iso-density lines running vertically), so when images digitized in the two different patterns of illumination are divided pixel-by-pixel the resulting ratio image encodes position in the beam. This information, combined with the pixel location, identifies the point in space of any surface illuminated and imaged onto the pixel. It is important that the relative position and orientation of the camera and slide projector remain unchanged between calibration and use of the device, that ambient light be handled properly, and that the camera response be linear. This range sensor produces a high resolution (512 horizontal x 480 vertical x 11 bits deep) range image with a registered intensity image of the scene in one minute. A second projector was recently added to this sensor.

A GE 2507 solid state video camera and two Kodak ektagraphic 3B slide projectors made up the sensor in these experiments. The work area, roughly 20 cm wide, 18 cm high and 25 cm deep, started 1 m from the camera, and the slide projectors were displaced about 78 cm from the camera. Placing projectors on both sides of the camera improved the range images by eliminating regions viewed by the camera but not illuminated by a projector. Range images were formed using the projectors separately, but calibrations were made so range images made using either projector could be directly combined: since the same camera is used with both projectors the images are registered. Images were digitized and most computations performed on a VICOM image processor. A VAX 750 controlled the VICOM using the interface program vsh [CH85], and also controlled the slide projectors.

Both projector - camera combinations were calibrated by making range images of a flat screen placed in the work area perpendicular to the camera axis at 10

measured distances. The range values were smoothed by averaging each pixel with its neighbors to reduce effects of noise in the camera response, then a cubic polynomial in range vs ratio was generated for each pixel, resulting in 5 images of calibration constants. These calibration images were used to convert ratio images to range images using

$$Z = A + r(B + r(C + rD)) \qquad (1)$$

where Z represents distance parallel to camera axis from a base plane at the front of the work area, r is the difference between the observed ratio and a center ratio R, and A,B,C,D, and R are images generated during the calibration. Eq. (1) is valid for every pixel, i.e., every letter has subscript i,j attached. Generation of the calibration images required about 30 minutes to get data and an hour of computation; the calibration remains good until a projector or the camera is moved. Transforming a ratio image to a range image is done on the VICOM image processor (with 12 bits for arithmetic) in a few seconds when the calibration images are in memory. The projectors and camera are bolted to a board so the unit as a whole can be moved (for experimental convenience) without disturbing the calibration. However, the projectors do not hold their positions perfectly, and in practice one projector can be adjusted slightly (so both sensors yield identical range measurements for an object) without causing serious error in the calibration. This same method of calibration and computation can be used with a range sensor that projects multiple bar patterns on a scene; if gray code ordering of bars is used the observed (temporal) bit pattern at any pixel is converted to an appropriate binary number with a look-up table before the computation is made. This procedure circumvents some distortion in camera and projection optics, and may be faster than the procedure of [SI87].

A range image is made by placing an object where it is illuminated by the projectors and viewed by the camera. The diaphragm is adjusted to let in as much light as possible without reaching maximum camera response anywhere. Images are made in the ambient light, with the first slide, and with the second slide. All images are corrected for non-linear camera response by subtracting the dark current (a substantial contribution with this camera) and using a look-up-table to replace each value with the linearized value[B87]. The ambient light is subtracted from each intensity image, then the ratio of these is formed and the ratio is converted to range at each pixel using Eq. (1). The process is then repeated with the second projector. Each range image is made in one minute: 20 sec. to make images (each image is the average of 8 digitizations to reduce noise), 14 sec to compute the ratio (special inversion algorithm to avoid losing bits), and 30 sec to compute the range (90% of which is spent reading calibration images from disk). The range images from the two projectors are combined into one image using data from whichever range image was formed from larger intensity values, since range values are more reliable in regions of greater intensity. Where the two images are expected to be of comparable quality both images are used, with a smoothly varying weighting function bridging the region of comparable intensity. Intensity images are combined by simply averaging them, a procedure which works well when shadows are not cast on the object.

With sufficient averaging the range sensor provides reliable data, but individual pixels under normal working conditions can exhibit considerable deviation (as much

as 0.5 cm) from expected values. The average range uncertainty due to random noise (due mostly to variations in camera output) is ±0.12 cm. This uncertainty can be reduced by increasing the number of images averaged, or by averaging spatially on the image surface (this blurs detail, of course). Systematic deviations of ±0.2 cm are present due to dust on and pinholes in the filter; in these experiments the projector was focussed outside the work area to spread these disturbances over an area of several square centimeters. Overall flatness and linearity were measured by averaging blocks of 16x16 pixels; the observed range values were linear in real range with a random variation of ±0.025 cm (error in screen placement is ±0.008 cm), and average deviation from flatness for a flat plane was ±0.03 cm. These measurements were made with a uniformly white matt screen. When abrupt changes in intensity occur (as at a change of reflectivity) deviations in range values are observed that seem to be caused by over- and under-shoots in camera electronics. However, as shown by the successful matching, the range sensor performs satisfactorily.

3.3. Data Reduction and Model Building

Model building in this implementation consisted of three phases: reducing each view to a set of points, successively matching appropriate sets of points to determine transformations for each view, and forming a model using the transformations and reduced data (curves). Reduction of a view to a set of points depends on the data available and on the object; however, it seems likely that nearly any object can be described for matching purposes by a relatively small set of points. The matching problem can be formulated in general terms as follows. Given N sets of points, each set reduced from a single view, find candidate views likely to have overlapping subsets of points. Generate an optimum order in which views should be matched. Validate true overlapping subsets. Match successively to form a complete model, and adjust for the accumulated matching error. While the final model may be just the points used for matching (or a representative set of them), it may be desirable to make a model consisting of lines or surfaces. In this case representative features can be formed from corresponding portions of transformed original data.

All algorithms have been developed to work with data that contains random variations at large and small scales. Reduction of complexity was not a primary factor in the design of these algorithms, although an attempt has been made to keep "n" small and operations efficiently coded. Data structures, as important as algorithms in determining performance, are described in the appendix.

Algorithms that were executed on the VICOM were written as chains of VICOM commands (vsh macros); algorithms executed on the VAX were written in C with the exception of the least squares matching routine, which was written by Micha Sharir in Fortran [SS85][BSSS86]. The algorithm of [AHB87] is the same. Algorithms given here are described using pseudo-code. Routine names are printed in bold face and described in the appendix.

DATA EXTRACTION:

For each view, after data acquisition:

VICOM holds composite intensity and range images
average the range image using a 3x3 low pass filter two times.
find jump edges in range image (use VICOM's Sobel Magnitude command)
find edges in intensity image (Diff. of Gaussians operator)
make mask of all intensity edges that are not range edges
multiply mask times range image
extract all curves > 120 pixels long on VAX (primedges)

DATA CONVERSION. Have lists of pixel locations and values that must be converted to locations in three-space. An extracted closed curve consists typically of 250 pixels and reduces to about 50 pixels. Each view contains about 11 curves, including both desired edges of disks and lines from the edges of the pots.

For every view
reduce data by averaging (pipecleaner)
convert all points to real space (edithcon)

DETERMINE TRANSFORMATIONS and MAKE MODEL:

for each view
sample curves to make point spacing uniform
for each curve
for all later curves
find points of closest approach (find_closest)
make valid ones into feature points (find_centroid)
for all pairs of feature points ("lines")
fill in data structure (process_lines)

build_match:
for each pair of views (do find_cand)
award points to possible corresponding curves
record average score as indication of overlap

generate Candidates[] list of consecutive views (plan_match)

for each successive pair of Candidates except last with first
produce feature points ("keypoints") (best_match())
solve for transformation (call_lstsq())
transform i+1st view to ith coordinate system (rotate_view())

distribute accumulated error

determine unique set of curves (model_correspondence, clean_correspondence)

for each unique curve
determine average curve from all representations (line_builder)

3.4. Performance

The present implementation of the automatic model builder constructs a model from 9 views (99 curves, containing 3527 points) in 1.1 minutes on a lightly loaded VAX 785. The time is roughly linear in the number of views, although some penalty is paid for the step which is quadratic in the number of views. In experiments using 18 views (198 curves, containing 7023 points) measurements of running time varied between 2.5 minutes to over 7 minutes depending on machine load. Changing program parameters also affected running time. Elapsed time can be roughly assigned to the various procedures as

8%	read in data from disk
11%	pre-process data
7%	**find_cand** for all pairs of views
65%	determine keypoints for each match (**best_match**)
7%	successive matching and final adjusting
2%	building model by averaging curves

Time used in **best_match** is taken primarily by the least squares procedure to check the many possible combinations of curves. The algorithm has not been optimized in any way, although attempts have been made to speed things up by throwing out obviously incompatible data before **call_lstsq**. More could probably be done in this direction. In addition, the least squares code, which has simply been taken from earlier work and plugged in, should be examined for unnecessary functions and cleaned up for this application. Changing MIN_COUNT from 1 to 10 improved the speed of this section by about 4% with no change in the final match. Changing MIN_COUNT from 1 to 20 sped up **best_match** by 20% with slight changes in the keypoints, but the final model was visually indistinguishable from the earlier model,

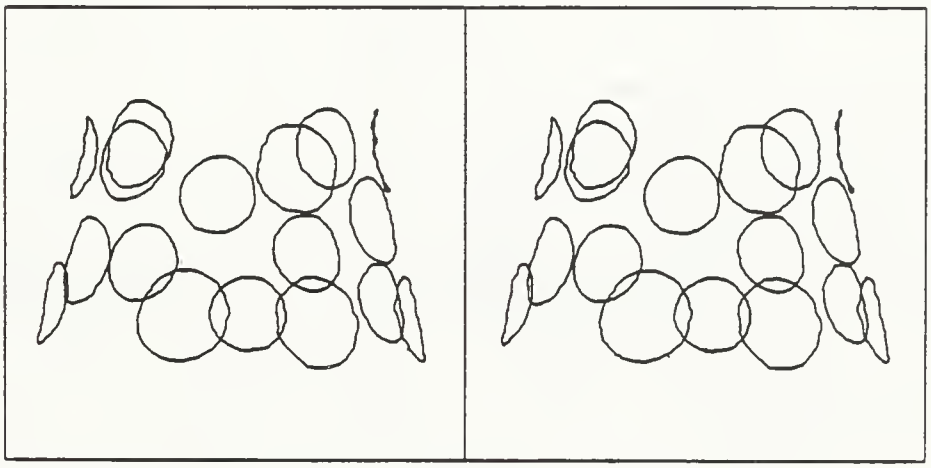


Figure 2. Stereo view of automatically constructed model. Curves represent edges of circular paper disks glued to an inverted flowerpot.

showing that good matching occurs even when some valid curves are not used.

Figure 2 shows a stereo view of the complete model, constructed from one example of each curve. Distortion of the circular disks by being wrapped about the conical surface is clearly visible. The model reflects the accuracy of the data available from the range sensor, since the final model of the object consists of original data merely rotated and translated in such a way as to get data from multiple views into a single coherent model.

The error in this model can be estimated from the error observed after all views had been successively matched but before this error was distributed, as in Fig. 3. An overlap of the first view with the last view showed a typical rotational displacement of about 11°. When distributed, this error corresponds to displacement

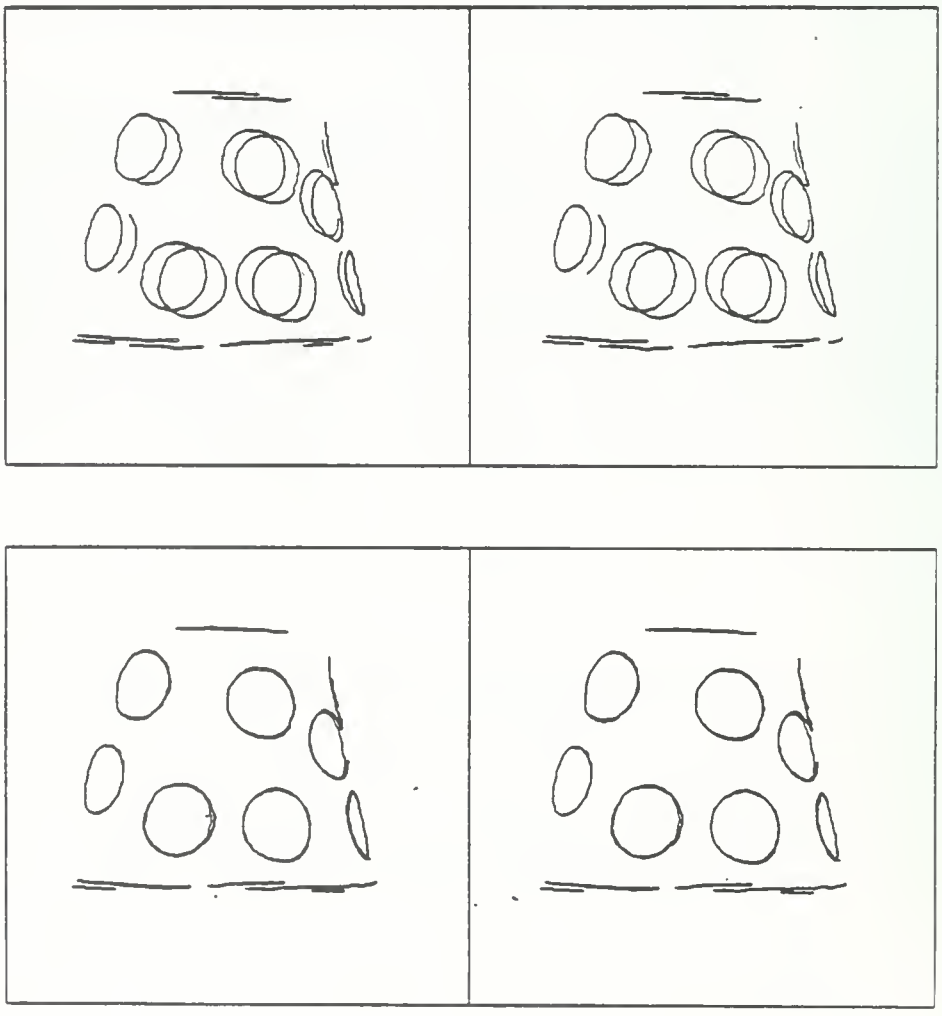


Figure 3. Views 6 and 7 after matcharound (top) and after adjustment (bottom).

of each curve by 2% of the diameter of its circular disk, thus 0.7 mm. In addition, examining models created from different choices of individual curves suggests that portions of curves representing the same object disk may differ in position by as much as 4 mm, though on average much less. Uncertainty in calibration parameters, particularly the angle of the camera field of view, was found to contribute greatly to observed match-around error. In addition, accurate work must explicitly take into account distortions present in any lens, and this was not done in these experiments.

4. Summary and Further Work

Presently implemented software and hardware permit the automatic construction of a 3D model from observations of a 3D object. The model builder differs from previous geometric model builders by using a model description based on curves in 3-space rather than on surfaces. It has performed extremely well, has provided valuable experience in model construction, and promises to generalize for use with many types of objects and features. The features used, outlines of patches of different reflectivity, have not been exploited in previous machine vision studies. The feature points extracted, points of closest approach, are novel and have proven reliable, even with simulated data much noisier than real data.

In one respect the present model builder is unnecessarily general, since it does not know which views contain overlapping regions, or any information about how the object location was changed between the various views. In practice it should be possible to make original images so that consecutive images would contain overlapping regions. However, determining an appropriate ordering of views takes negligible time in this implementation. In other respects the present implementation could be more general. Determination of corresponding feature points makes use of the underlying curves, and thus the implementation of **best_match** is not suited for general sets of points, although the basic algorithm could be used. At present only one type of feature is used, and other significant curves (e.g., the edges of the pot which are incidentally extracted and remain in the data set) are not used.

This work will be extended and improved. It is most important to develop additional types of features points and to use them effectively. For example, edges of objects should be used, and other geometric features. Since many manufactured objects contain circles and straight lines these should be included as features in some way. Objects from the "blocks world" should also be handled, but present software does not know about vertices. In algorithms for robot vision time is more important than mathematical elegance and generality, and thus algorithms that can extract and use a small set of quickly identifiable features are highly desirable. The most expensive routines, **best_match** and the least squares determination of transformations, will be examined and made more efficient. Experiments will be made in which objects are presented to the range sensor so consecutive views can be matched in known order to yield a complete model of the object, and in which objects will have multiple closed cycles of views.

References

[AHB87] K.S. Arun, T.S. Huang, and S.D. Blostein, "Least-squares fitting of two 3-D point sets" *IEEE Transactions on Pattern Analysis and Machine Intelligence* **PAMI-5**, pp 698-700, 1987.

[B86] C.M. Bastuscheck, "Finding representative points of closest approach for noisy curves", Robotics Report #83, Robotics Activity, New York University, 1986.

[B87] C.M. Bastuscheck, "Correction of video camera response using digital techniques", Optical Engineering (in press); also Robotics Report #105, Robotics Activity, New York University, 1987.

[BHH83] R.D. Bolles, P. Horaud, M.J. Hannah, "3DPO: a three-dimensional part orientation system", in *Proc. 8th International Joint Conf. on Artificial Intelligence*, pp. 1117-1120, 1983.

[BS84] C.M. Bastuscheck, J.T. Schwartz, "Preliminary implementation of a ratio image depth sensor" Robotics Report #28, Robotics Activity, New York University, 1984.

[BS86] C.M. Bastuscheck, J.T. Schwartz, "Experimental implementation of a ratio image depth sensor", in *Techniques for 3-D Machine Perception*, A. Rosenfeld, ed. North Holland, 1986.

[BSSS86] C.M. Bastuscheck, E. Schonberg, J.T. Schwartz, and M. Scharir, "Object recognition by three-dimensional curve matching", Int. J. Intell. Systems 1 pp. 105-132 (1986).

[CH85] D. Clark, R. Hummel, "VSH users' manual: an image processing environment", Robotics Report #19, Robotics Activity, New York University, Third edition, 1985.

[CMST87] C.I. Connolly, J.L. Mundy, J.F. Stenstrom, and D.W. Thompson, "Matching from 3-D range models into 2-D intensity scenes", *Proc. 1st Int. Conf. Computer Vision*, pp. 65-72 (1987).

[FH83] O.D. Faugeras, M. Hebert "A 3-D recognition and positioning algorithm using geometrical matching between primitive surfaces", in *Proc. 8th International Joint Conf. on Artificial Intelligence*, pp. 996-1002, 1983.

[Fea83] O.D. Faugeras, F. Germain, G. Kryze, J.D. Boissonnat, M. Hebert, J. Ponce, E. Pauchon, N. Ayache, "Towards a flexible vision system", in *Robot Vision*, A. Pugh, ed, pp. 129-142. IFS Ltd and Springer Verlag, 1983.

[GL84] W.E.L. Grimson, T. Lozano-Pérez, "Model-based recognition and localization from sparse range or tactile data", *The International Journal of Robotics Research* 3, pp. 3-35 (1984).

[HB84] P. Horaud, R. Bolles, "3DPO's strategy for matching three-dimensional objects in range data", *Proceedings of the International Conference on Robotics*, IEEE New York, pp. 78-85, 1984.

[Kn87] J. Knapman, "3D model identification from stereo data", *Proc. 1st Int. Conf. Computer Vision*, pp. 547-551 (1987).

[Ko86] P. Kofakis, "Recognition and localization of objects using sparse data", *Proc. Conf. on Computer Vision and Pattern Recognition*, pp. 647-650 (1986).

[L85] D.G. Lowe, "Visual recognition from spatial correspondence and perceptual organization", in *Proc. 9th International Joint Conf. on Artificial Intelligence*, pp. 953-959, 1985.

[OS83] M. Oshima, Y. Shirai "Object recognition using three-dimensional information", *IEEE Transactions on Pattern Analysis and Machine Intelligence* **PAMI-5**, pp 353-361, 1983.

[P83] M. Potmesil, "Generating models of solid objects by matching 3d surface segments", in *Proc. 8th International Joint Conf. on Artificial Intelligence*, pp. 1089-1093, 1983.

[SI87] "Range-imaging system utilizing nematic liquid crystal mask", *Proc. Ist Int. Conf. Computer Vision*, pp. 657-661 (1987).

[SS85] J.T. Schwartz, M. Sharir, "Identification of partially obscured objects in two and three dimensions by matching of noisy *Characteristic Curves*", Robotics Report #46, Robotics Activity, New York University, 1985.

[TM87] D.W. Thompson and J.L. Mundy, "Three-dimensional model matching from an unconstrained viewpoint", *1987 IEEE Int. Conf. on Robotics and Automation* pp. 208-220 (1987).

[VA86] B.C. Vermuri and J.K. Aggarwal, "3-D model construction from multiple views using range and intensity data", *Proc. Conf. on Computer Vision and Pattern Recognition*, pp. 435-437 (1986).

Appendix: Description of Algorithms

primedges:
input: image containing all zeros except for single pixel wide curves which contain range values (integers between 1 and 2047)
output: list of continuous curves represented by triplets of (i,j,value)
algorithm:
start at upper left corner
read consecutive rows until value > 0 is encountered
store this location
follow curve to end, making values negative
return to starting point, and follow curve to other end
if curve has more than 120 pixels, write out from one end to other
return to stored location and continue to read rows to end of image
Notes: curve following routine tolerates breaks 1 pixel wide.

pipecleaner:
input: set of curves as written by primedges
output: similar data, but fewer points and less noise
algorithm:
For each curve,
remove extremely low values (due to VICOM hardware error)
For each curve,
average 5 consecutive range values
write result using coordinates of middle pixel
advance 5 pixels and repeat sequence

if end is close to start, indicate that curve is closed

edithcon:
input: data from pipecleaner in machine units
parameter file CALIBRATION.DATA
output: curves expressed in realspace x,y,z coordinates.
algorithm:
for every point
convert range to z coordinate with camera at origin
convert horizontal pixel to x using horizontal field of view
convert vertical pixel to y using vertical field of view
subtract offset of workspace from camera so workspace starts at 0
notes: workspace starts distance "d0" cm from camera.
there are "cunit" cm per 2000 integers of unconverted range.

find_closest:
input: two curves in 3 space represented by points equally spaced along the curves
output: indices of the points at which the two curves are closest
smallest distance of any of these points from an end of its curve

algorithm:

```
for each point on curve 1
        for each point on curve 2
                find distance squared (d2)
                if d2 < min_distance  record indices and new min_distance
```

notes: With smooth, quasi-convex curves a more efficient algorithm found closest points much faster. However, with real data (and noise) that algorithm has proven less robust than this one. The requirement that points of closest approach be a minimum distance from the ends of curves eliminates spurious feature points caused by partially observed curves.

find_centroid:

input: curve represented by sampled points, and index of a point

output: average of coordinates of the index point and n points on each side

notes: n is determined as balance between many points (better representation) and availability of long curves. n, 7 here, depends indirectly on the sampling interval, 0.3 cm here.

process_lines:

input: pointer to a view

output: for each piece, a linked list of lines having one end on that piece

algorithm:

```
for each piece
        find every line to or from this piece
        sort by increasing length
        for each line
                make P_rel
                enter pointer to line
                adjust pointer to P_rel
```

notes: a piece is a curve (see discussion at start of description of data structure). The linked lists help to make the relations between curves easier to find in later searches for corresponding curves in different views.

build_match: calls many routines, see text.

find_cand:

input: pointers to two views (A and B)

output: value indicating similarity between two views; and
a stored array indicating correspondence between all pairs of curves

algorithm:

```
for every line of A
   for every line of B
        if similar length
                add to SCORE[i][j] for i,j = end piece indices
                add bonus to SCORE[i][j] if very similar in length
                for every other line of A
```

for every other line of B
if lines of A have common vertex piece
if lines of B have common vertex piece
add to SCORE[][] if angles are similar
add bonus if angles are very similar
save array SCORE on a stack for later use
sum all elements of SCORE
divide by (# curves in A) x (# curves in B) for final value
notes: lengths within 0.3 cm got 1 point, within 0.1 cm 3 more;
angles within 0.1 radian got 3 points, within 0.003 5 more.

plan_match:
input: array RESULTS[A][B] containing values from **find_cand** for all views A and B
output: list of views in the order in which they can best be successively matched
algorithm:
generate array RANK[][] from RESULTS[][] by replacing highest value in each row by 1, next highest by 2, etc until CUT entries are made. CUT = 6 here.
start solution with view (row index) having highest value in RESULT[][]
while legal continuations can be generated
for each of up to CUT possible solutions carried forward
find up to CUT different continuations by adding 1 view to either end
from these (up to CUTxCUT) solutions choose the CUT best cumulative scores
notes: All scoring used the RANK[][] array, which normalizes peculiarities observed in RESULTS scoring. That low numbers correspond to high scores (but rank = 0 is not used) complicates coding. Including scoring of next neighbors helped stabilize growth. This algorithm does not include closed cycles in any form; this is a needed improvement.

best_match:
input: pointers to two views
SCORE[][] array stored for these views by **find_cand**
output: complete sets of corresponding feature points ("keypoints") for the two views
algorithm:
find all likely pairs corresponding curves (i.e. SCORE[][] > MIN_SCORE)
for each pair of corresponding curves
for each other pair of corresponding curves
for each remaining pair of corresponding curves
if valid corresponding vertex sets, award points (VOTES)
reduce set of all correspondences to unique correspondences with most VOTES
determine keypoints
update global array of corresponding curves mod_piece_corr[][]
notes: a vertex set is three curves which are connected by two lines. In scoring, a vertex correspondence gets 2 points, the ends 1 each. Keypoints are those for which a pair of curves in A and corresponding curves in B have feature points defined. The algorithm is expensive because it is $O(n^3)$ in the number of corresponding pairs of curves, and because transformations between vertex sets are found (call_lstsq).

Requiring corresponding legs to have lengths within 0.5 cm of each other eliminated many matches, and when all three correspondences had at least 12 votes the procedure did not check for good correspondence, reducing execution time further. Increasing MIN_SCORE improves performance slightly. It is important to have as many keypoints as possible. However, bad or doubtful points should never become part of keypoints.

call_lstsq:
input: 2 sets of points in 3 space
output: rotation and translation required to bring the sets into closest juxtaposition
the distance (error) between the juxtaposed sets
algorithm:
least squares solution for rotation and translation are determined separately [BSSS86][SS85][AHB87].

rotate_view:
input: pointer to a view
transformation parameters
output: rotate and translate all data of view (original points, and feature points)

Distribute accumulated error:
input: closed cycle of N matched views, with keypoints
possible offset between first and last views
output: closed cycle of matched views with error evenly distributed.
any offset is shrunk by a factor N
algorithm:
corresponding keypoints K of view 0 and view N-1 should overlap, but do not
using keypoints, determine transformation T_f to take K_{N-1} to K_0 (call_lstsq)
for every set of keypoints K
form a set of shadow keypoints: $S = T_f K$
form final keypoints as weighted combination: $F = \frac{N-i}{N}K + \frac{i}{N}S$
determine transformation from K to F (**call_lstsq**)
transform entire view to final location and orientation (**rotate_view**)
notes: The shadow keypoints exist in locations which the keypoints would have if the matching had started at view 0 and progressed in the opposite order.. All shadow views connect nicely except for the probable gap between S_0 and S_1. The algorithm essentially averages the two matchings.

model_correspondence:
On each call of **best_match** the global array mod_piece_corr[][] is updated. Rows represent different curves of the model, columns are successive views (in order of matching).
1st call: piece numbers of corresponding curves are placed in first two columns, using as many rows as needed
ith call: if piece has been observed, new piece number is entered in ith column on that row, otherwise piece number is entered in ith column of first empty row.
last call: (signaled by reuse of first candidate): clean up. Many last observed curves may

be the same as first observed curves. Rows of such curves are combined.

clean_correspondence:
Further reduces mod_piece_corr array by removing curves which are not closed curves. This is specific to this experiment, where all curves can be observed as closed curves. Also, centroids of closed curves are found, and those rows are combined whose curves are within 1 cm of each other.

line_builder:
input: several complete representations of a closed curve in the same coordinate system
output: one curve representing the curve
algorithm:
Several are under test.
Simplest is to pick one curve, e.g. the first one encountered (which was done here)
Being tested:
translate each input curve so centroids are coincident (may not be needed)
determine most distant point from the common centroid
divide up the space around the centroid using a cubic array BOXES[][][]
put all points of the curves into BOXES, like hashing, but keeping spatial coherence
use boxes to determine which points are close in space
follow occupied boxes through space,
averaging close points (of several curves) for output points.

The following code set up the data structure for these experiments. Points, lines, p_rels, pieces and objects all had basic data stored in stacks. This data structure developed from one begun by Edith Schonberg, and certain features of it are not used much in model building, e.g. object names, dates, tags. In the text "pieces" are consistently called "curves", and "objects" are called "views", as the earlier names are not presently informative.

```
/* POINT is the data type for points.
 *
 *      x, y, z         are the real coordinates of the POINT.
 */
struct point_desc
{
        float           x;
        float           y;
        float           z;
};
typedef struct point_desc       POINT;

/* LINE is the structure representing two points of closest approach of
```

```
 * two pieces in one object.
 *      direction is unit vector representing end2 - end1
 *      end1 is aggregated centroid of piece1 , and similarly end2
 */
struct line_desc
{
        int                     piece1;
        int                     piece2;
        float                   length;
        POINT                   *direction;
        POINT                   *end1;
        POINT                   *end2;
};
typedef struct line_descLINE;

struct p_rel_desc
{
        LINE                    *lines;
        struct p_rel_desc       *next;
};
typedef struct p_rel_desc P_REL;


/* PIECE is the structure representing a curve of an object.
 */
struct piece_desc
{
        int                     npoints;
        POINT                   *points;
        P_REL                   *p_rels;
};
typedef struct piece_desc       PIECE;

/* OBJECT is the structure representing an observed object.
 *
 *      name            the name of the object.
 *      desc            description of OBJECT.
 *      tag             tag is used to discriminate among objects.
 *      date            date object is entered into catalog.
 *      npieces         number of pieces in an object.
 *      pieces          pointer to list of pieces for object.
 *      nlines          number of lines associated with the object
 *      lines           pointer to start of lines in array Lines.
 */
struct obj_desc
{
        char            name[MAX_LINE];
        char            desc[MAX_LINE];
```

```
        char        tag;
        char        date[26];
        int         npieces;
        PIECE       *pieces;
        int         nlines;
        LINE        *lines;
};
typedef struct obj_desc OBJECT;
```

NYU COMPSCI TR-321 c.2
Bastuscheck, C Marc
Automatic model builder for
object recognition.